SEGELN LERNEN MIT **SARSA**

PRÄSENTIERT VON SVEN FRITZ UND MARTIN ZAKARIAN KHENGI

GRUNDLAGEN ADAPTIVER WISSENSSYSTEME

ALLGEMEINE INFORMATIK (M.SC.)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

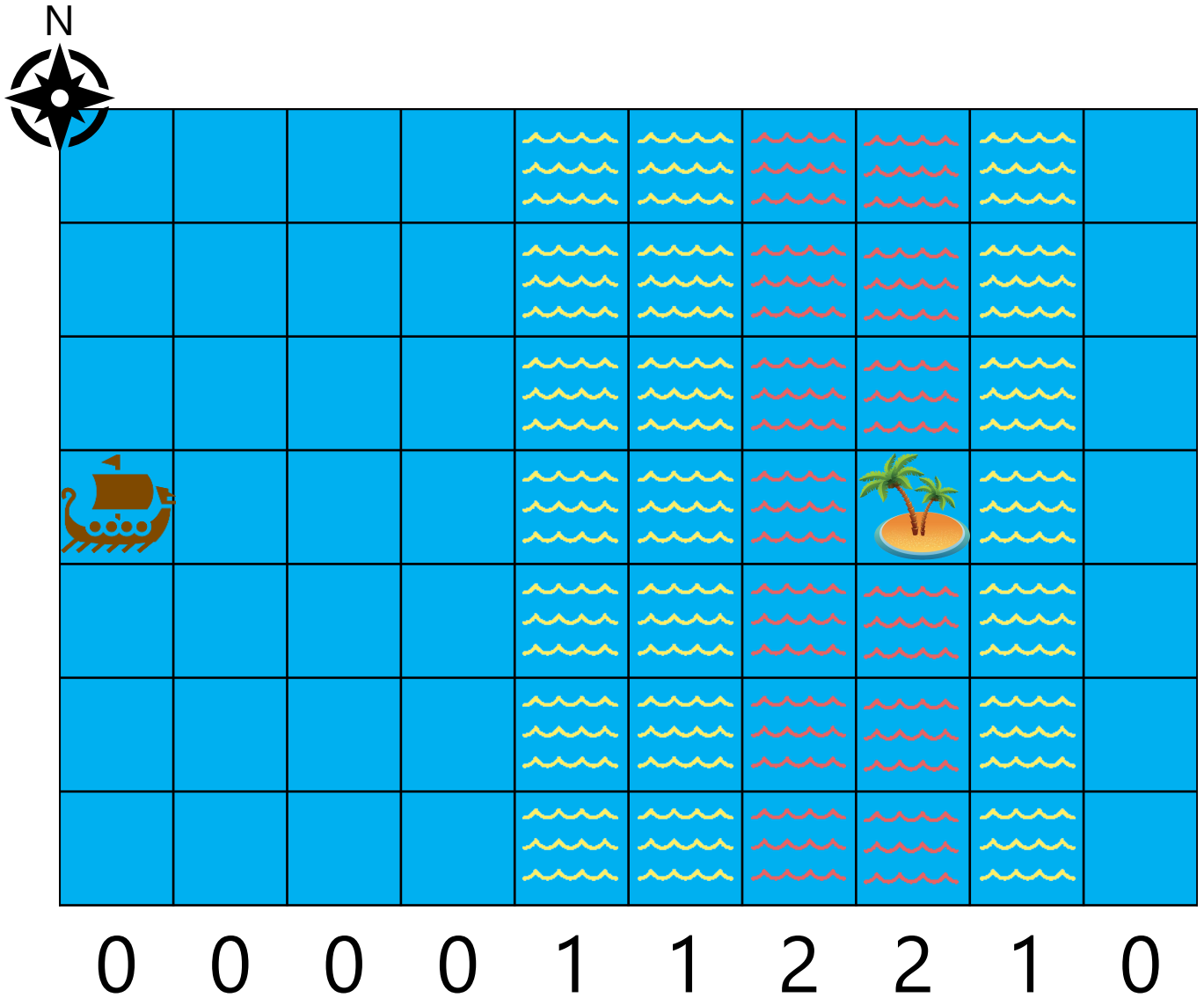
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

GLIEDERUNG 2

1. Aufgabenstellung
2. SARSA Algorithmus
3. Parameter
4. Implementierung
5. Demonstration
6. Ergebnisse

AUFGABENSTELLUNG 3

- Reinforcement Learning
- State–action–reward–state–action → SARSA
- Bestmögliche Strategie finden
- Umgebung ist für Agent unbekannt
- Interaktion mit der Umgebung
- Episodische Lernschritte
- Stationäre Strategie
- Deterministische und stochastische Betrachtung

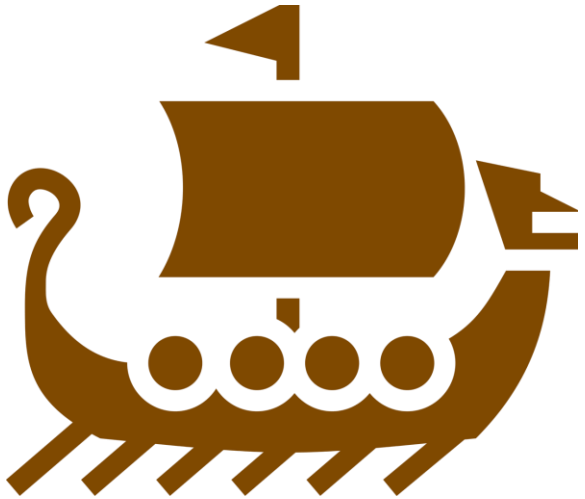


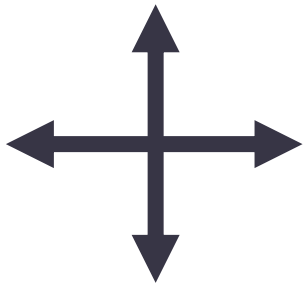
AUFGABENSTELLUNG

- See als Gitterwelt
- 10 x 7 Felder
- S = Startzustand
  - An- und Ablegestellen des Boots
- G = Zielzustand
  - Terminalzustand
- Nördlicher Wind
- SKP-Problem
- MDP = {T, S, A, p, r}

AUFGABENSTELLUNG 5

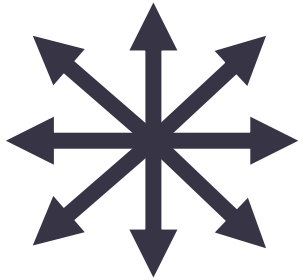
- Der Agent und die zu betrachtenden Aktionsmengen





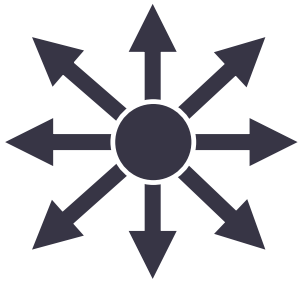
Normal

$$A_{No} := \{n, s, w, o\}$$



König

$$A_{Kö} := A_{No} \cap \{nw, no, sw, so\}$$



König+

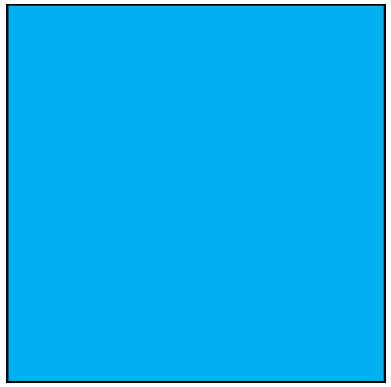
$$A_{Kö^+} := A_{Kö} \cap \{verweilen\}$$

AUFGABENSTELLUNG 6

- Beschreibung der Belohnungen

Insel

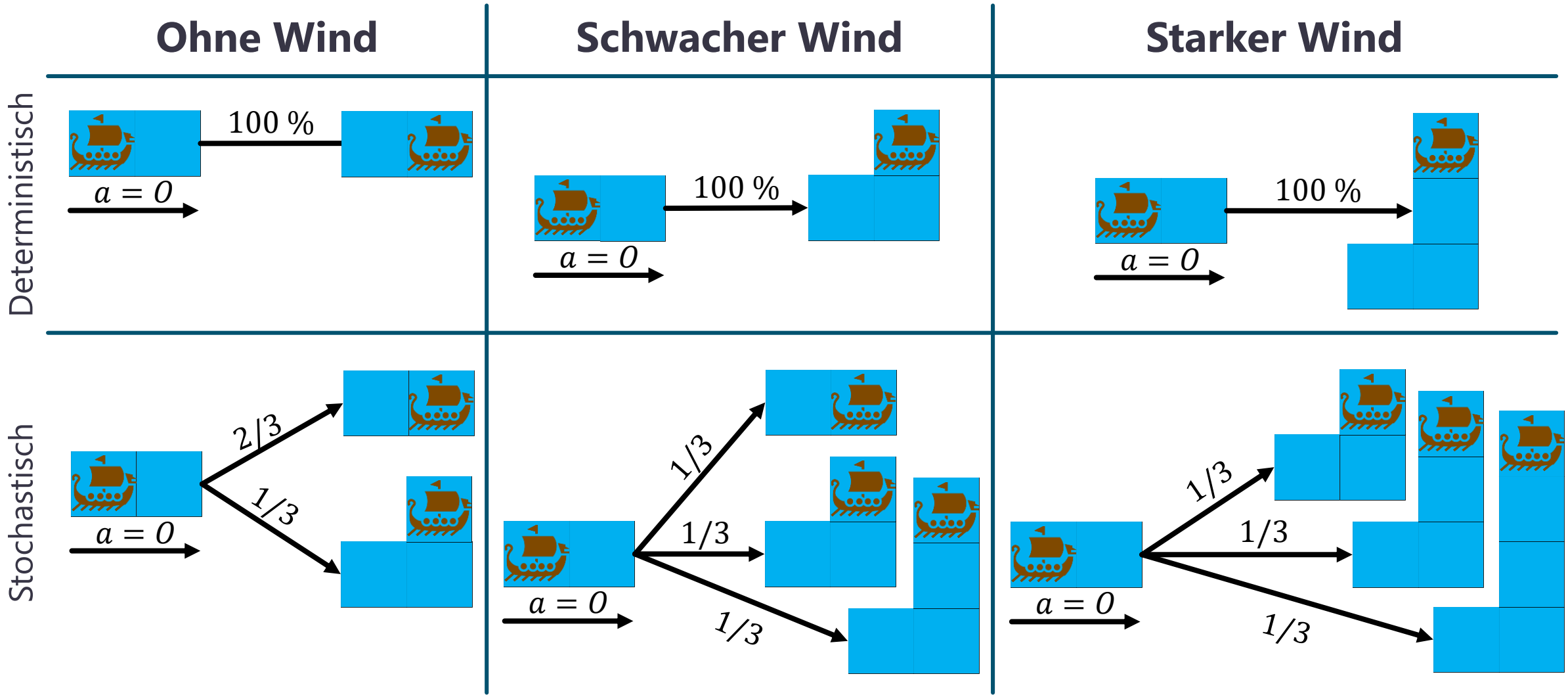
- Belohnung → 100
- Terminalzustand



See

- Belohnung: -1

AUFGABENSTELLUNG



SARSA ALGORITHMUS

- Pseudocode für SARSA

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

- Q-Value Update nach SARSA

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

PARAMETER

- Anzahl der Episoden → 5.000 Episoden
- Diskontierung → $\gamma = 1.0$
- Lernrate → $\alpha = 0.1$
- Strategie → $\varepsilon - greedy = 0.1$

IMPLEMENTIERUNG

Listing 1.1. SARSA Algorithm

```
def update(episodes=5000, show_steps=True, train=True, save=True):
    if train:
        for episode in range(episodes):
            # reset environment
            s1 = session.reset()

            # choose action based on policy
            a1 = agent.choose_action(str(s1))

            while True:
                # do action and get new state and its reward
                s2, r, done = session.step(a1)

                # choose action based on policy
                a2 = agent.choose_action(str(s2))

                # start learning SARSA algorithm
                agent.learn(str(s1), a1, r, str(s2), a2)

                # set new state as root for next iteration
                s1 = s2
                a1 = a2

                # break loop if terminal state is reached
                if done:
                    break
...
```

IMPLEMENTIERUNG

Listing 1.2. Action Decision Function

```
def choose_action(self, observation):
    self.check_state_exist(observation)
    # action selection
    if np.random.rand() > self.epsilon:
        # choose best action
        state_action = self.q_table.loc[observation, :]
        state_action = state_action.reindex(
            np.random.permutation(state_action.index)) # some actions
                have same value
        action = state_action.idxmax()
    else:
        # choose random action
        action = np.random.choice(self.actions)
    return action
```

IMPLEMENTIERUNG

Listing 1.3. Definition strong Winds

```
...
if s in self.strongWinds:
    if self.stochastic:
        temp = np.random.choice(np.arange(1, 4),
                                p=[(1 / 3), (1 / 3), (1 / 3)])
        if temp == 1:
            if s[1] + base_action[1] > UNIT * 3:
                base_action[1] -= UNIT * 3
            elif s[1] + base_action[1] > UNIT * 2:
                base_action[1] -= UNIT * 2
            elif s[1] + base_action[1] > UNIT:
                base_action[1] -= UNIT
        elif temp == 2:
            if s[1] + base_action[1] > UNIT * 2:
                base_action[1] -= UNIT * 2
            elif s[1] + base_action[1] > UNIT:
                base_action[1] -= UNIT
        elif temp == 3:
            if s[1] + base_action[1] > UNIT:
                base_action[1] -= UNIT
    else:
        if s[1] + base_action[1] > UNIT * 2:
            base_action[1] -= UNIT * 2
        elif s[1] + base_action[1] > UNIT:
            base_action[1] -= UNIT
...
```

IMPLEMENTIERUNG

Listing 1.4. Q-Value Update SARSA

```
class Sarsa(RL):

    def __init__(self, actions, alpha=0.1, gamma=1, epsilon=0.1):
        super(Sarsa, self).__init__(actions, alpha, gamma, epsilon)

    def learn(self, s, a, r, s_, a_):
        self.check_state_exist(s_)
        q_predict = self.q_table.loc[s, a]
        if s_ != 'goal':
            # next state is not terminal
            q_target = r + self.gamma * self.q_table.loc[s_, a_]
        else:
            # next state is terminal
            q_target = r

        # sarsa q-update update
        self.q_table.loc[s, a] += self.alpha * (q_target - q_predict)
```

	0	1	2	3
[5.0, 269.0]	-4.455.310.963.688.950	-4.377.242.221.583.820	-0.49964174703862396	-4.407.236.377.723.770
[5.0, 357.0]	-3.727.629.628.507.530	-36.818.200.578.482.100	-3.714.627.765.739.080	-3.671.152.124.675.330
[93.0, 269.0]	-3.708.087.321.350.050	-36.561.311.362.888.000	8.648.896.531.890.870	-363.782.058.403.764
[5.0, 181.0]	-4.220.788.678.227.780	-42.138.280.530.123.500	-4.164.871.628.229.730	-420.662.539.807.009
[93.0, 181.0]	-39.600.874.137.669.200	-38.924.602.987.846.800	-3.839.247.738.795.020	-3.915.494.771.127.290
[5.0, 445.0]	-3.172.859.080.526.740	-3.134.209.111.587.480	-3.181.863.147.956.130	-3.298.040.236.441.340

DEMONSTRATION LERNEN UND ANWENDEN

GRUNDLAGEN ADAPTIVER WISSENSSYSTEME

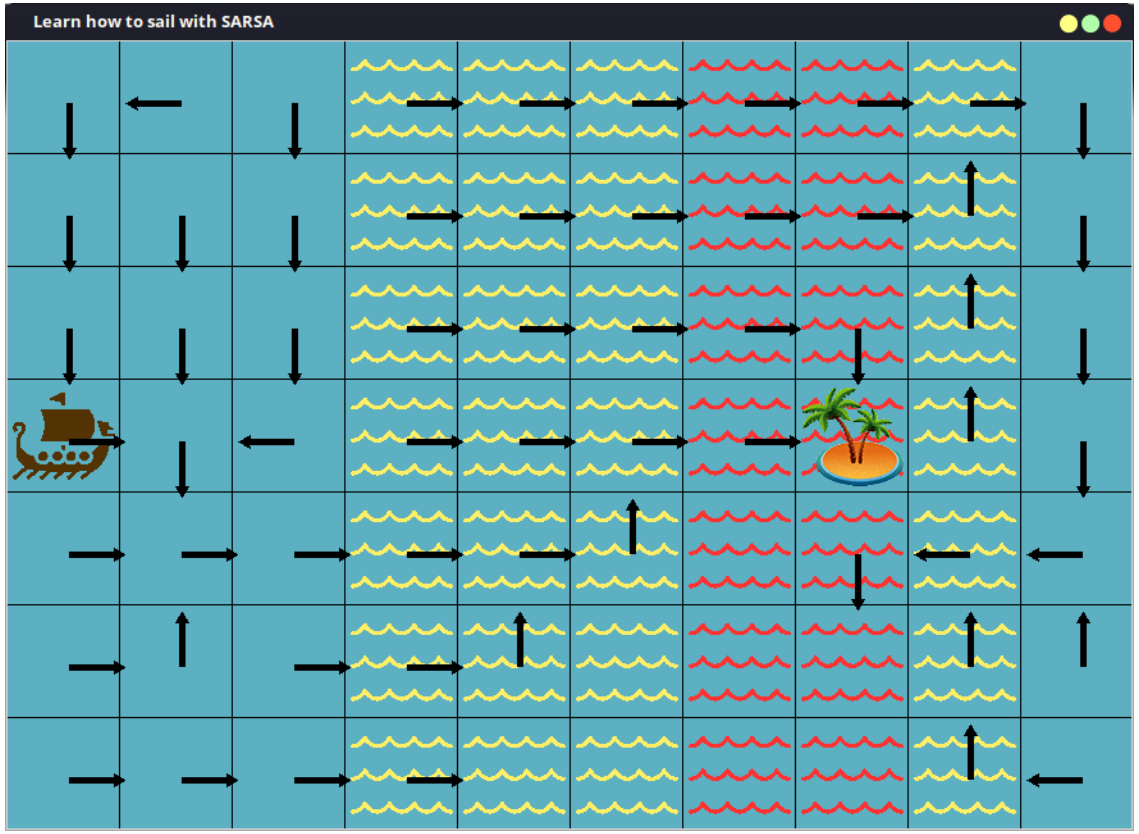
ALLGEMEINE INFORMATIK (M.SC.)

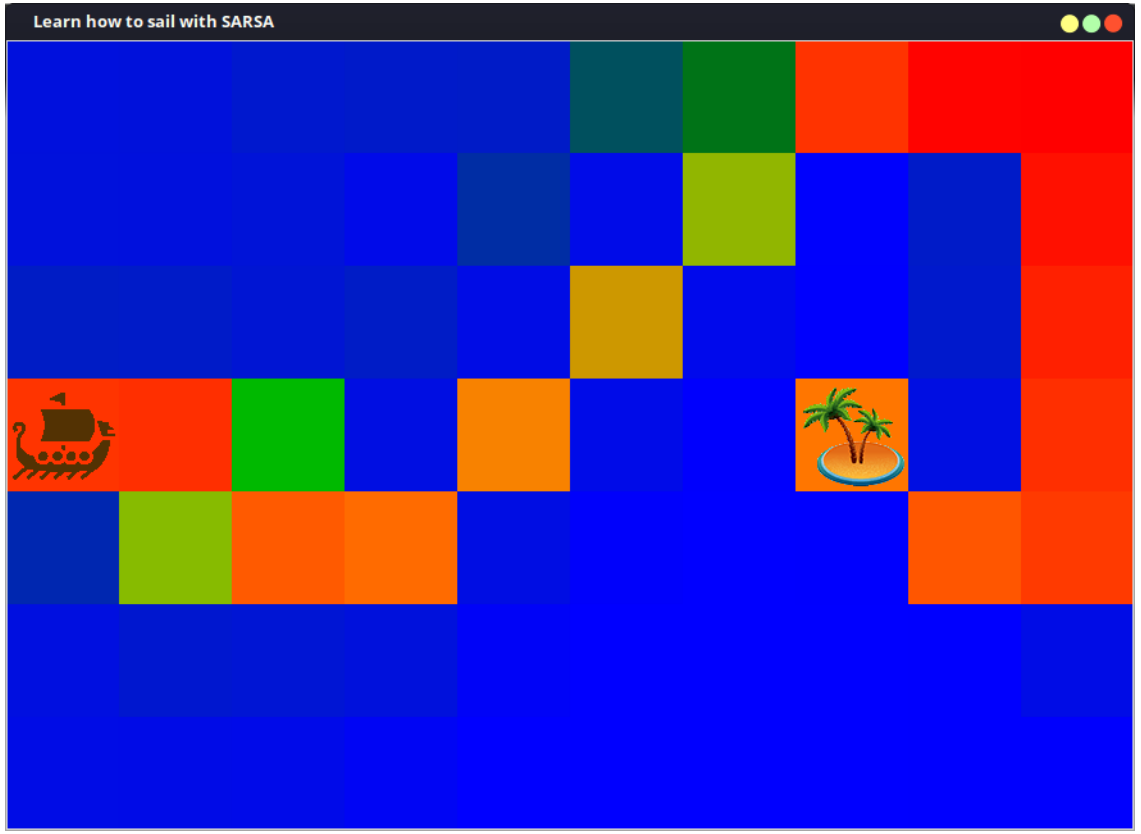
FRANKFURT UNIVERSITY OF APPLIED SCIENCES

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ERGEBNISSE

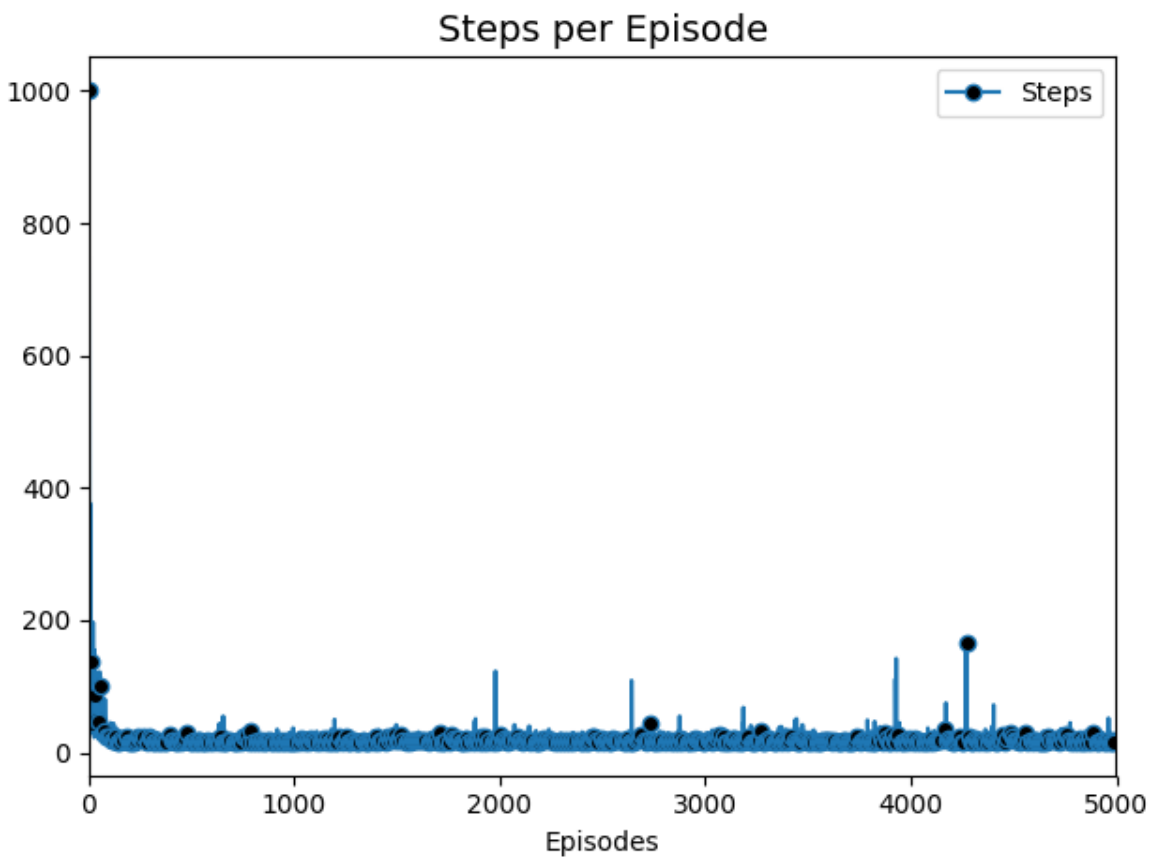
Aktionsmenge: Normal Umgebung: Deterministisch

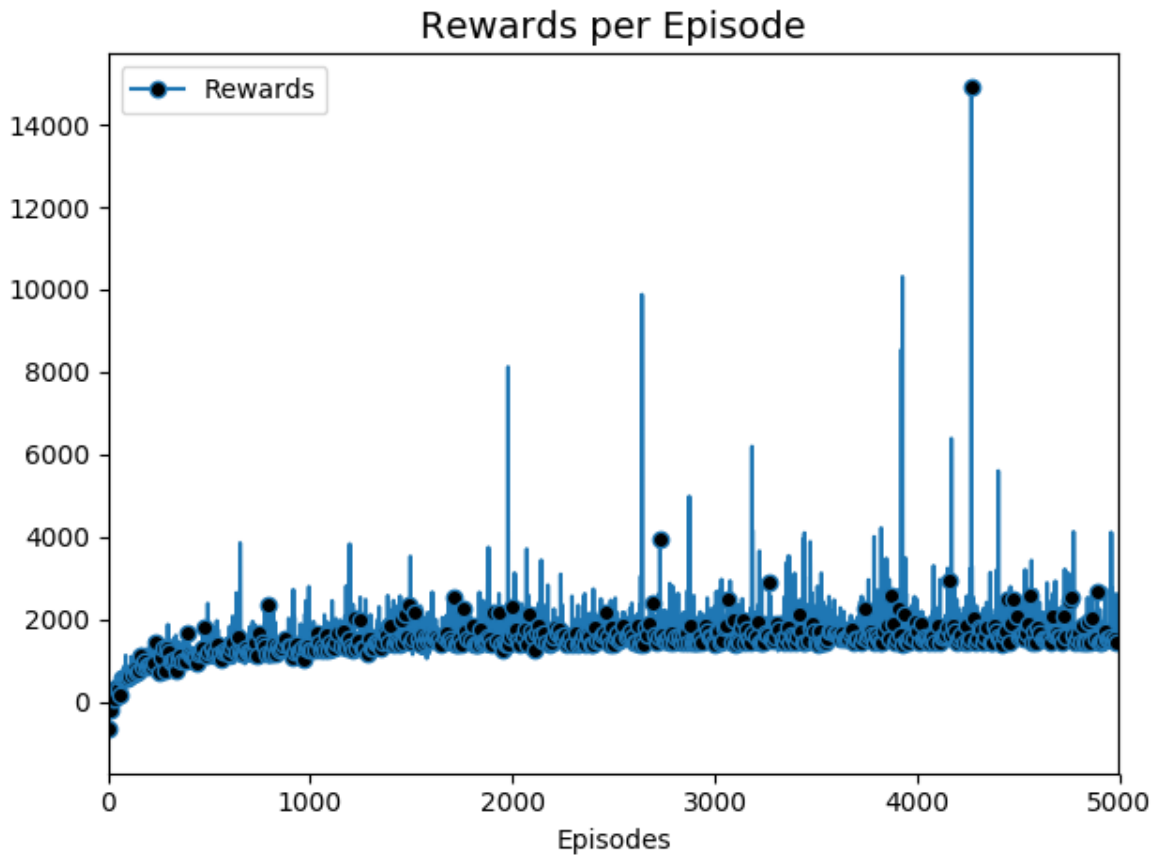




ERGEBNISSE

Aktionsmenge: Normal Umgebung: Deterministisch

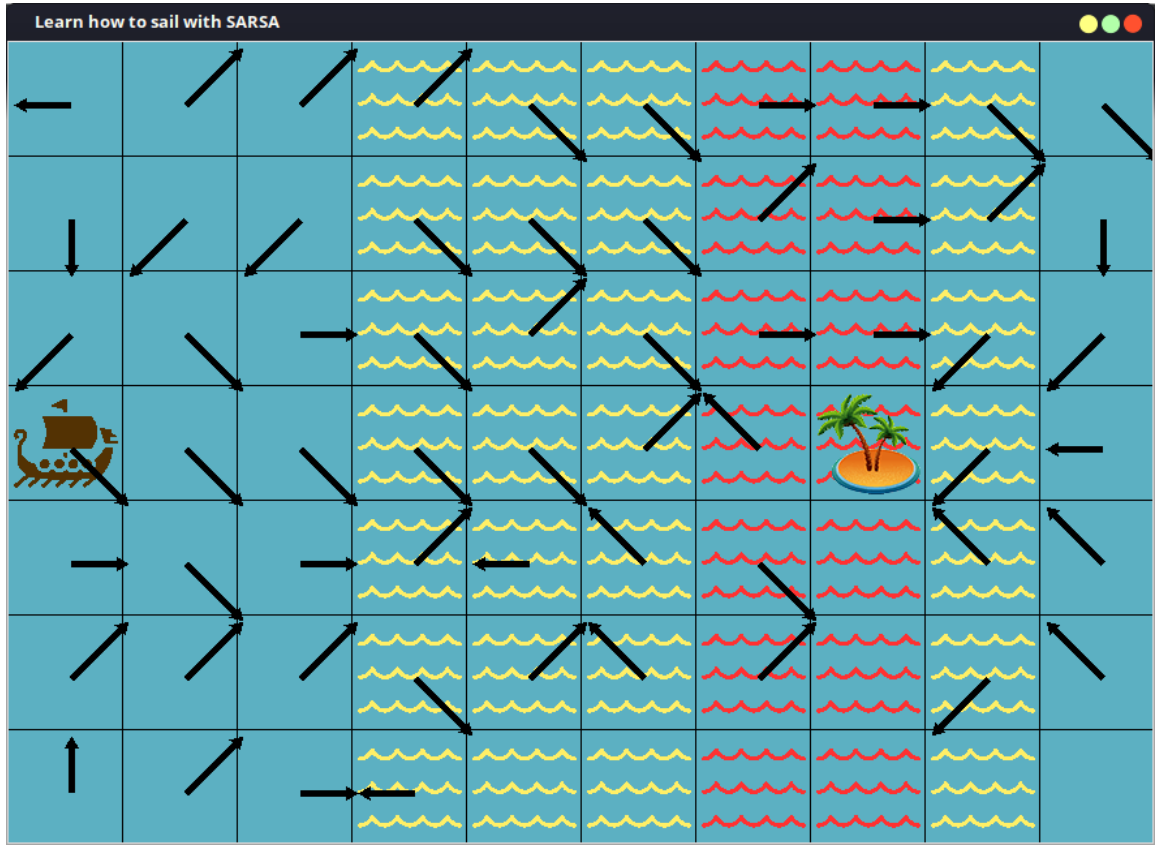




ERGEBNISSE

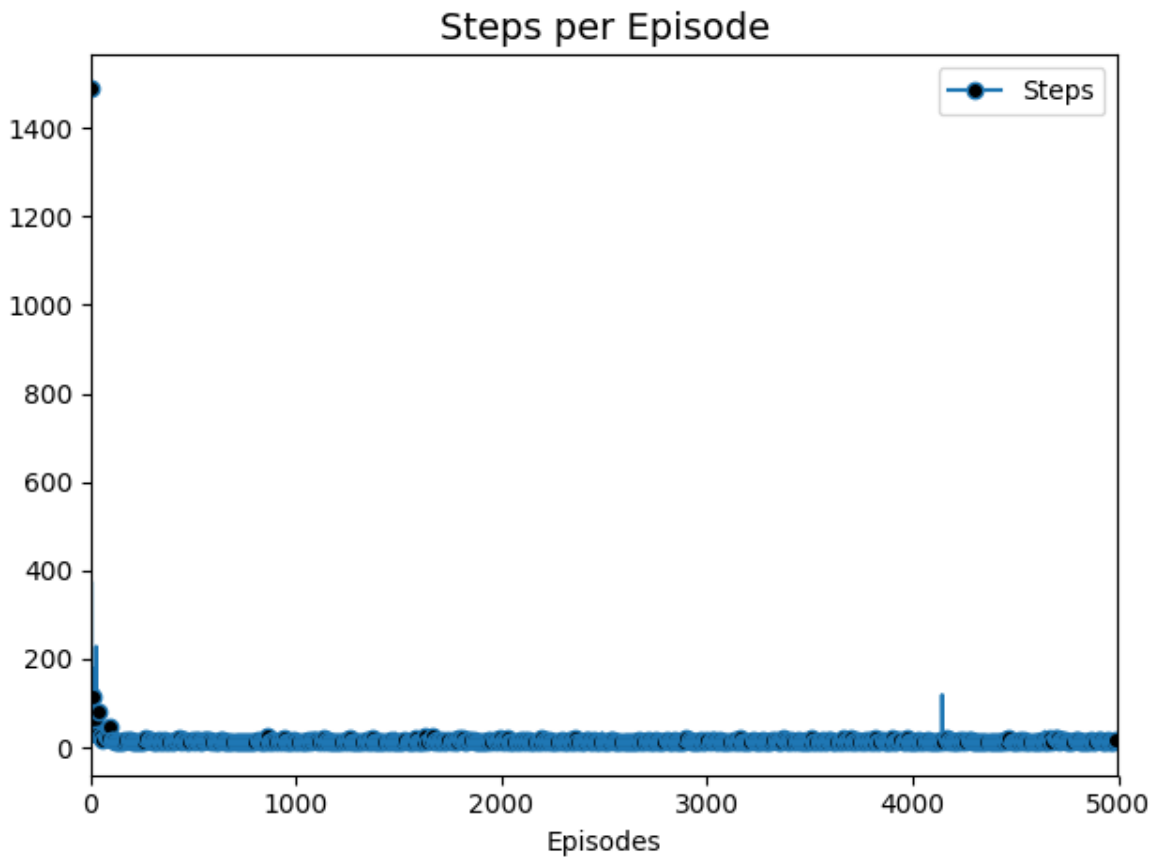
Aktionsmenge: König

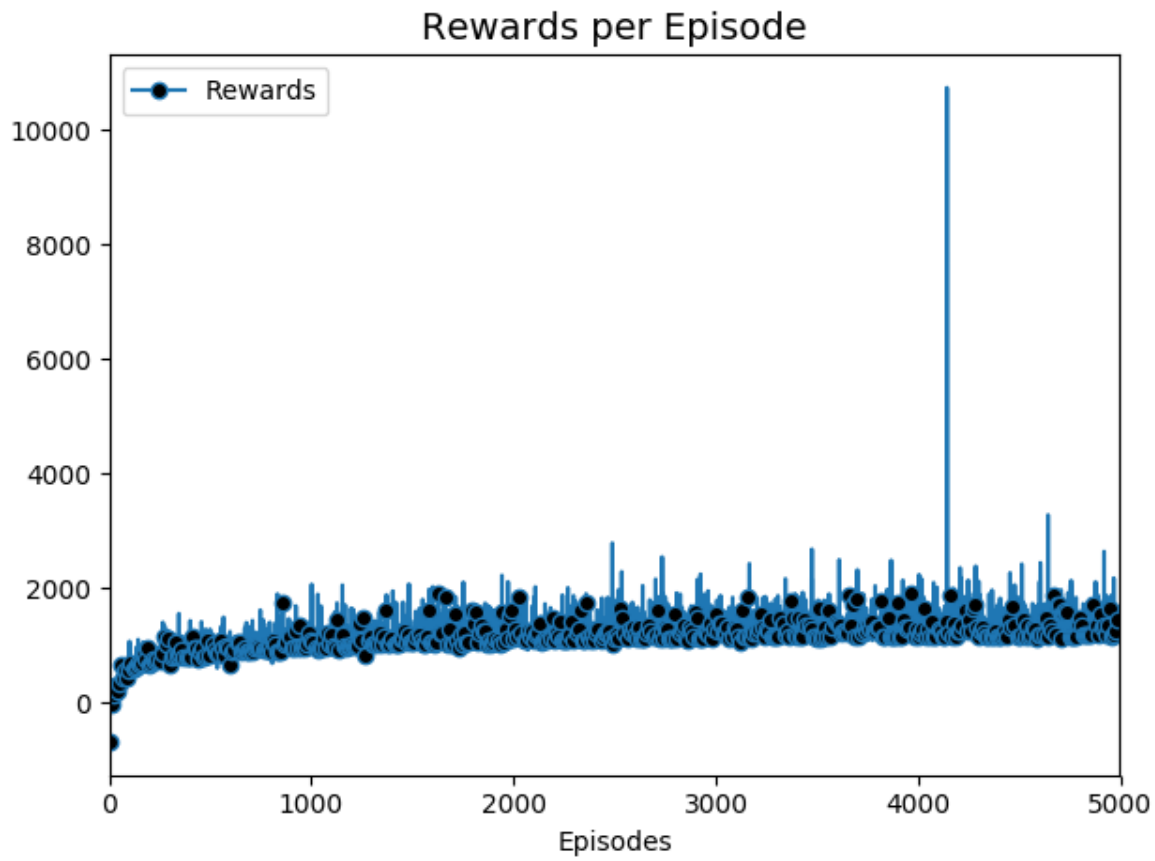
Umgebung: Deterministisch



ERGEBNISSE

Aktionsmenge: König Umgebung: Deterministisch

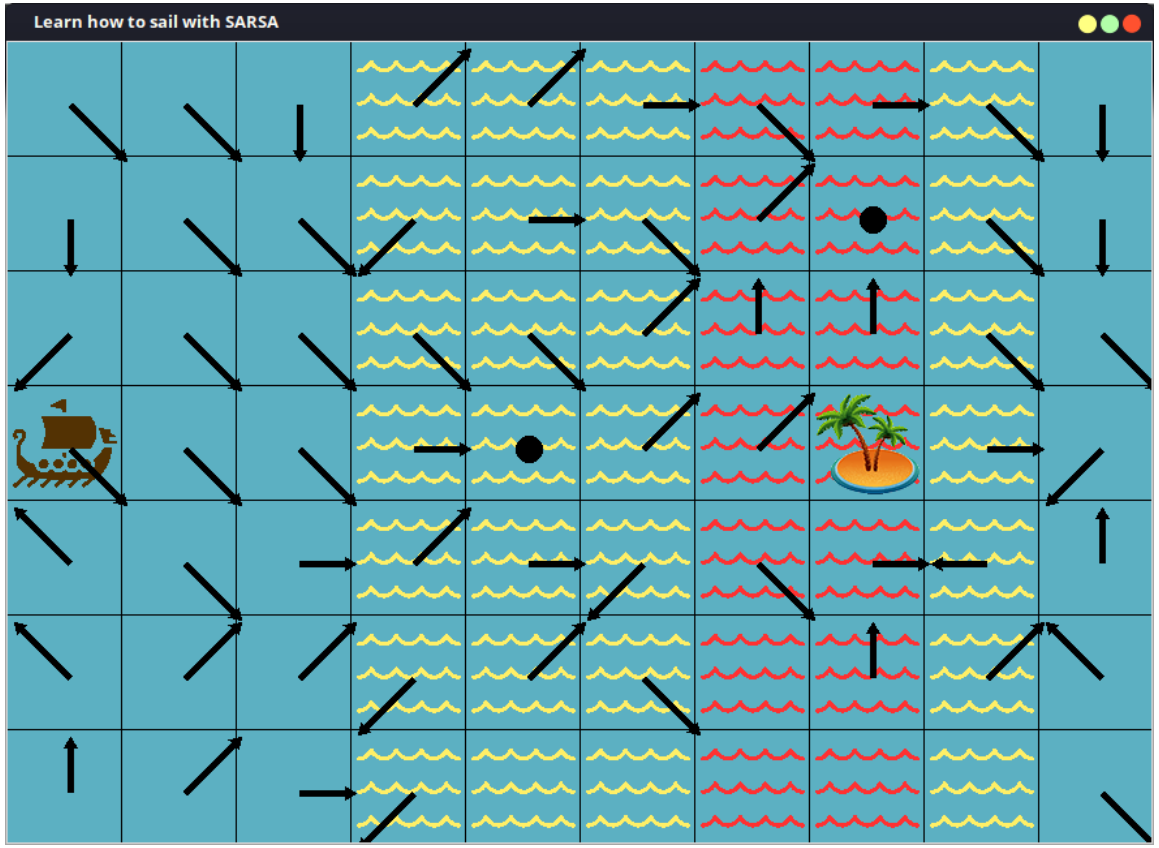


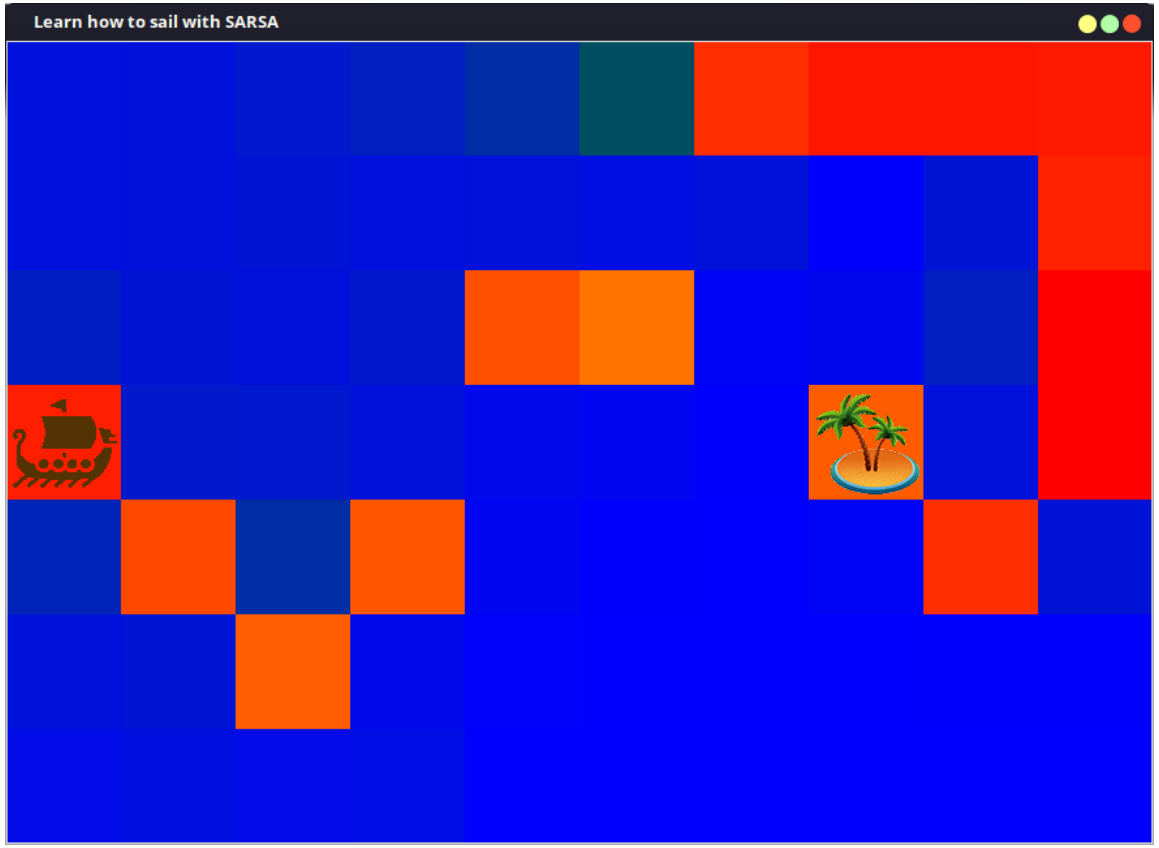


ERGEBNISSE

Aktionsmenge: König+

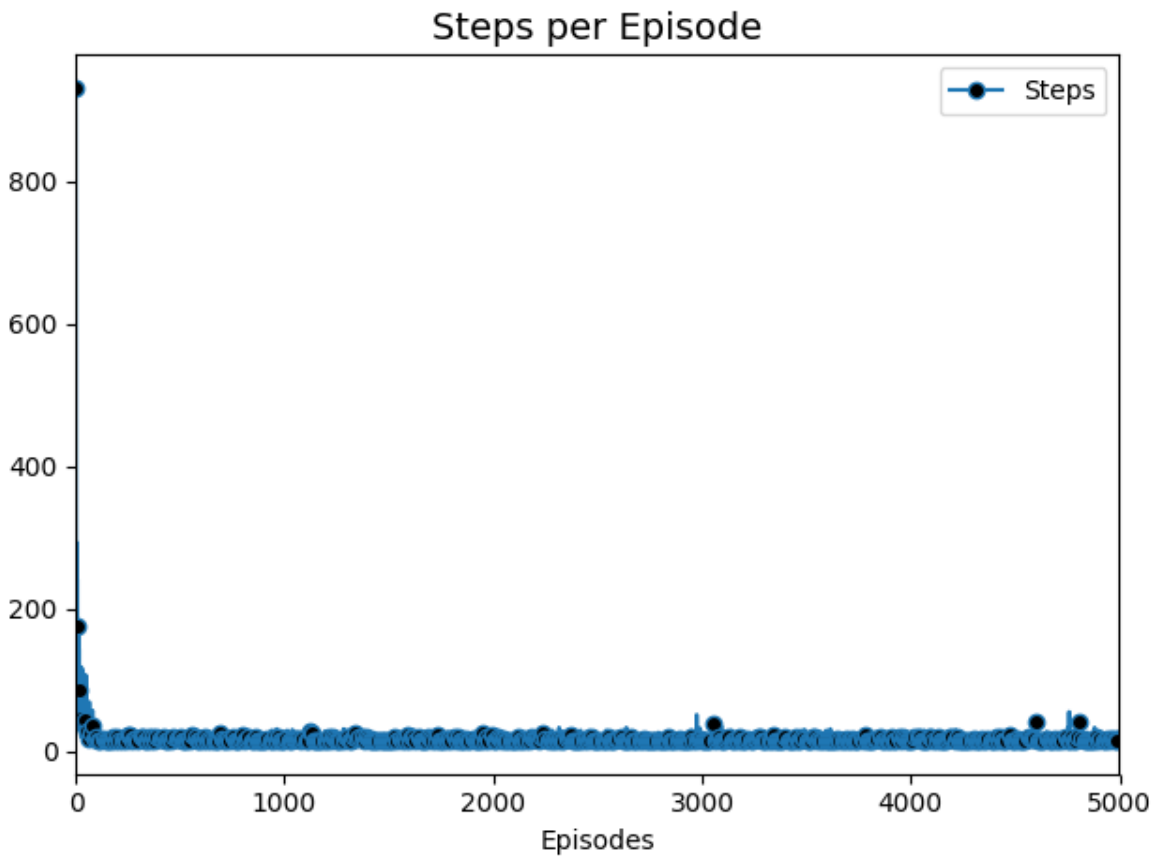
Umgebung: Deterministisch

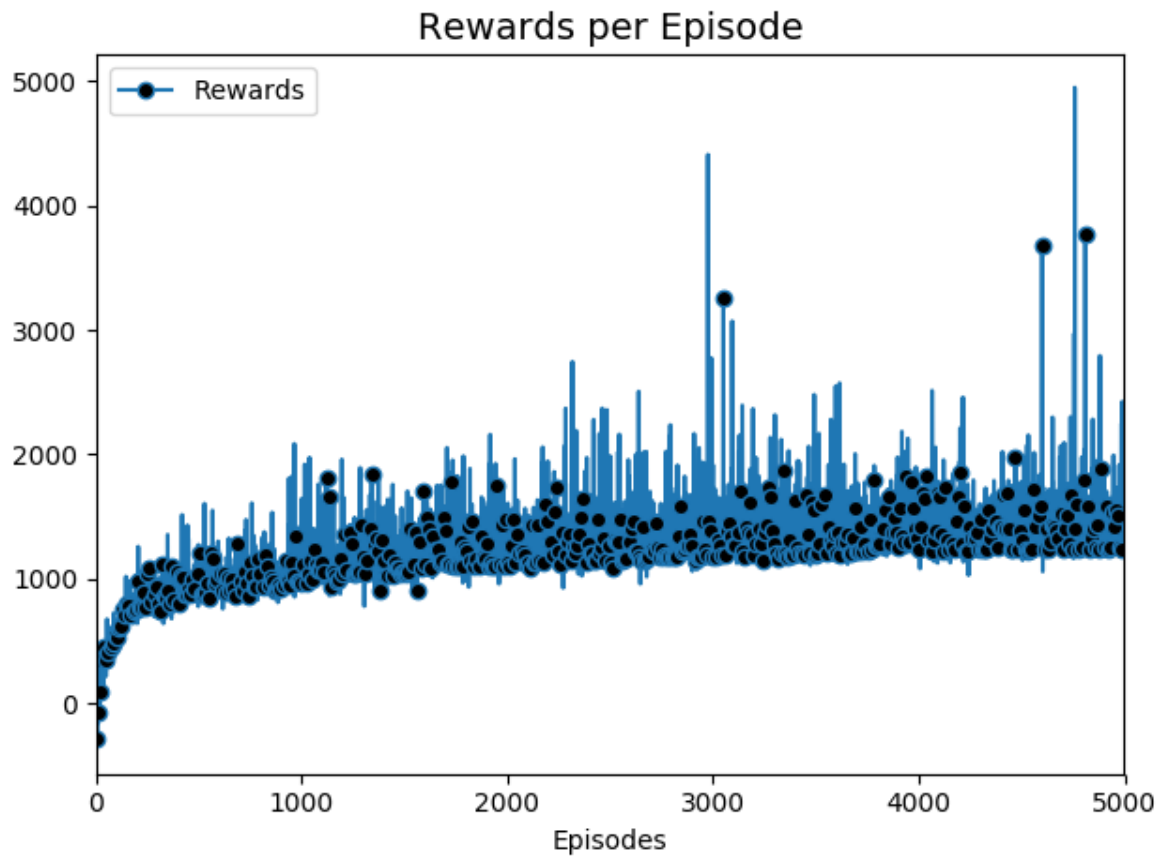




ERGEBNISSE

Aktionsmenge: König+ Umgebung: Deterministisch



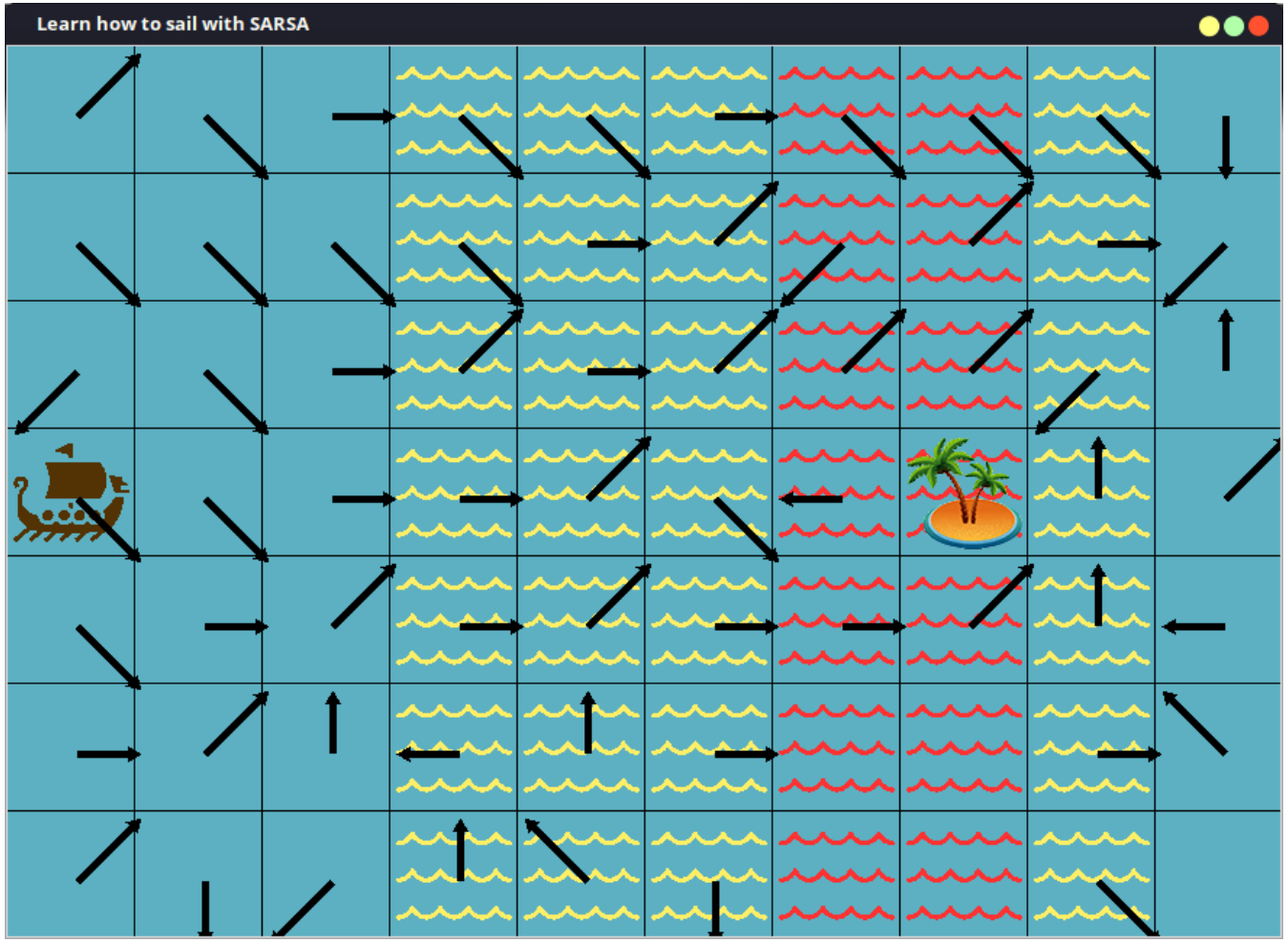


ERGEBNISSE

Aktionsmenge: Normal Umgebung: Stochastisch

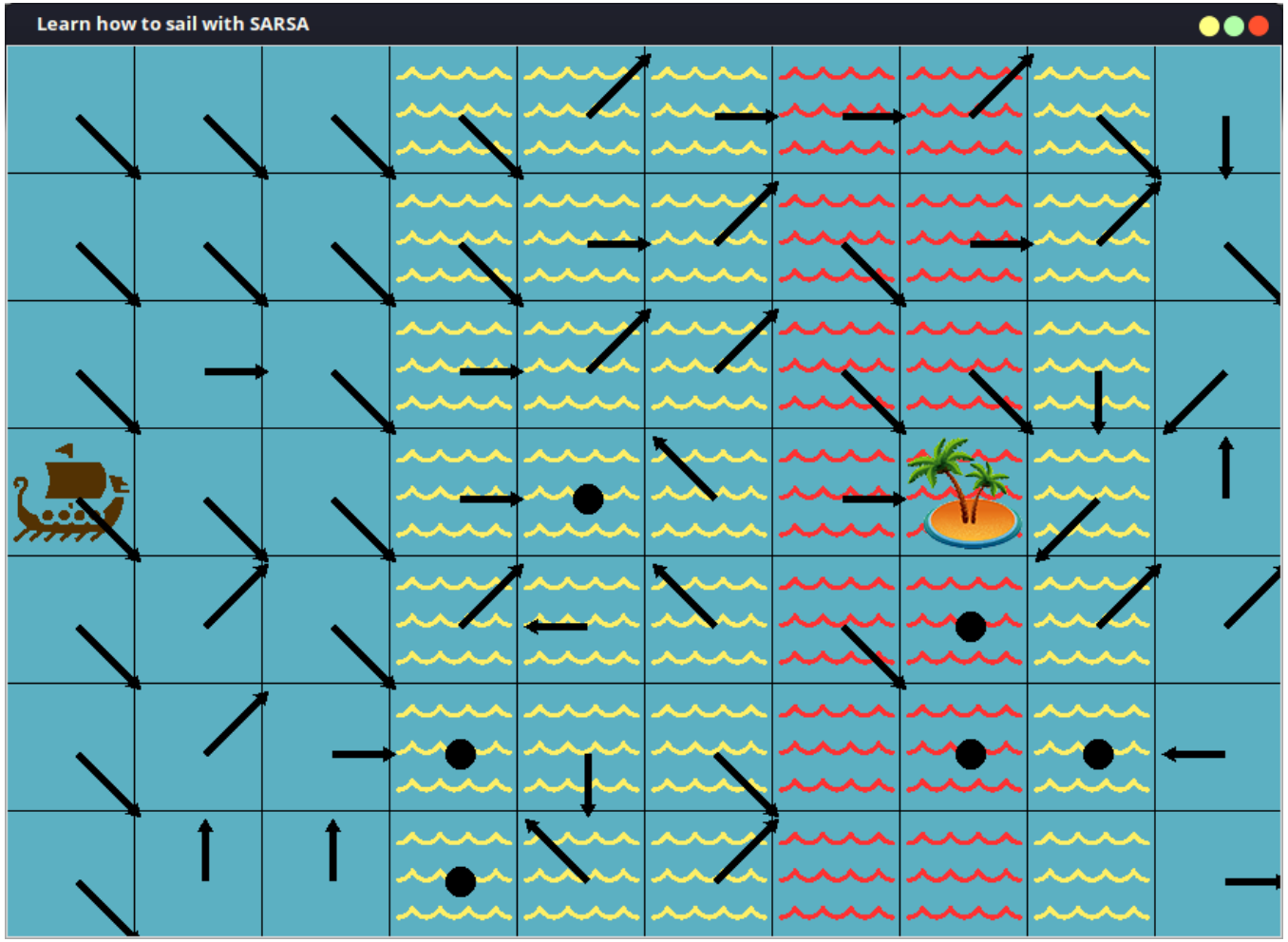
ERGEBNISSE

Aktionsmenge: König Umgebung: Stochastisch



ERGEBNISSE

Aktionsmenge: König+ Umgebung: Stochastisch

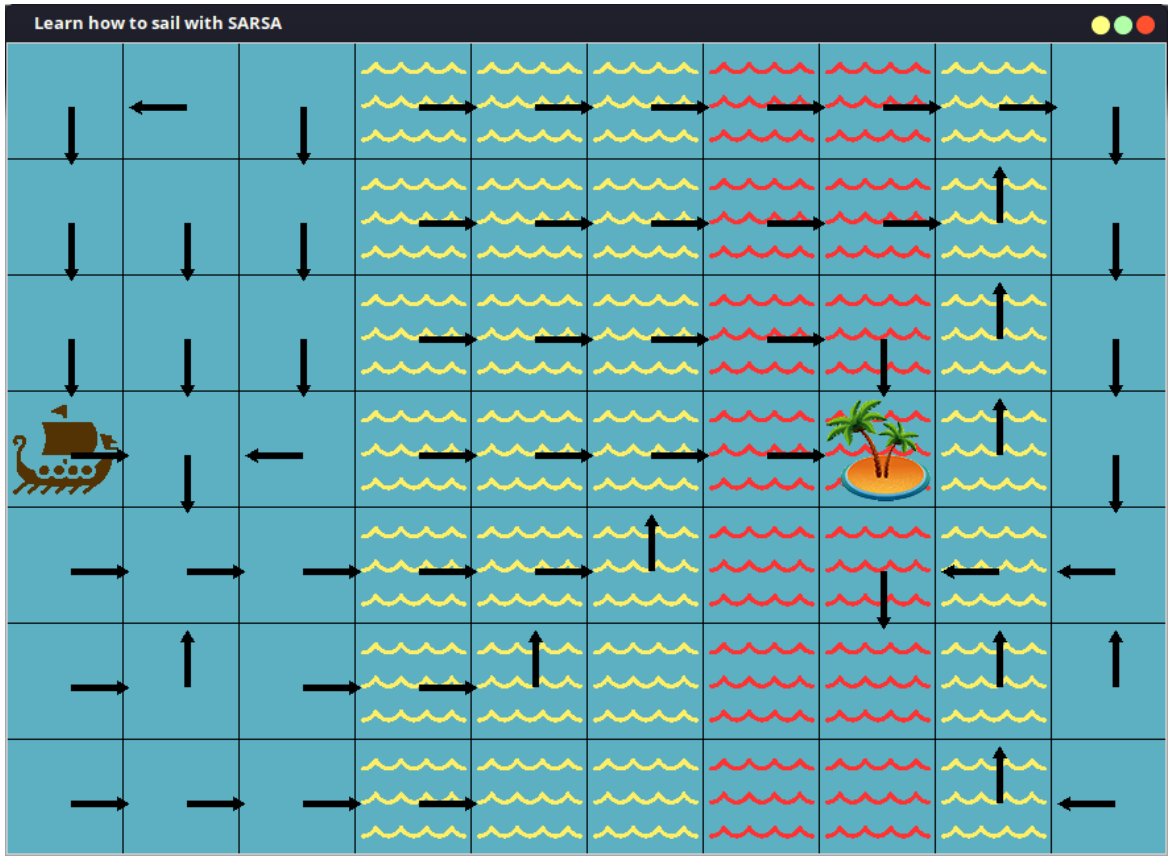


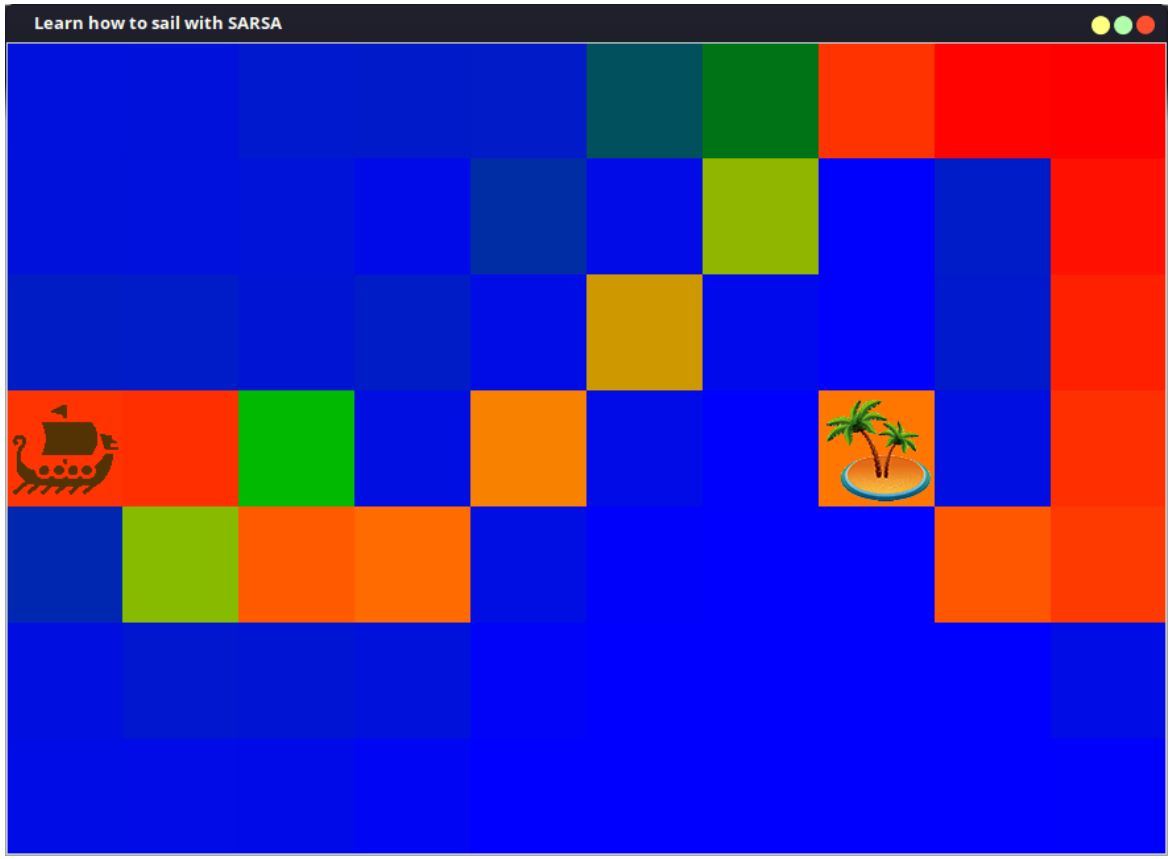
ERGEBNISSE

Aktionsmenge: Normal

Umgebung: Deterministisch

Beobachten von ε = 0.1



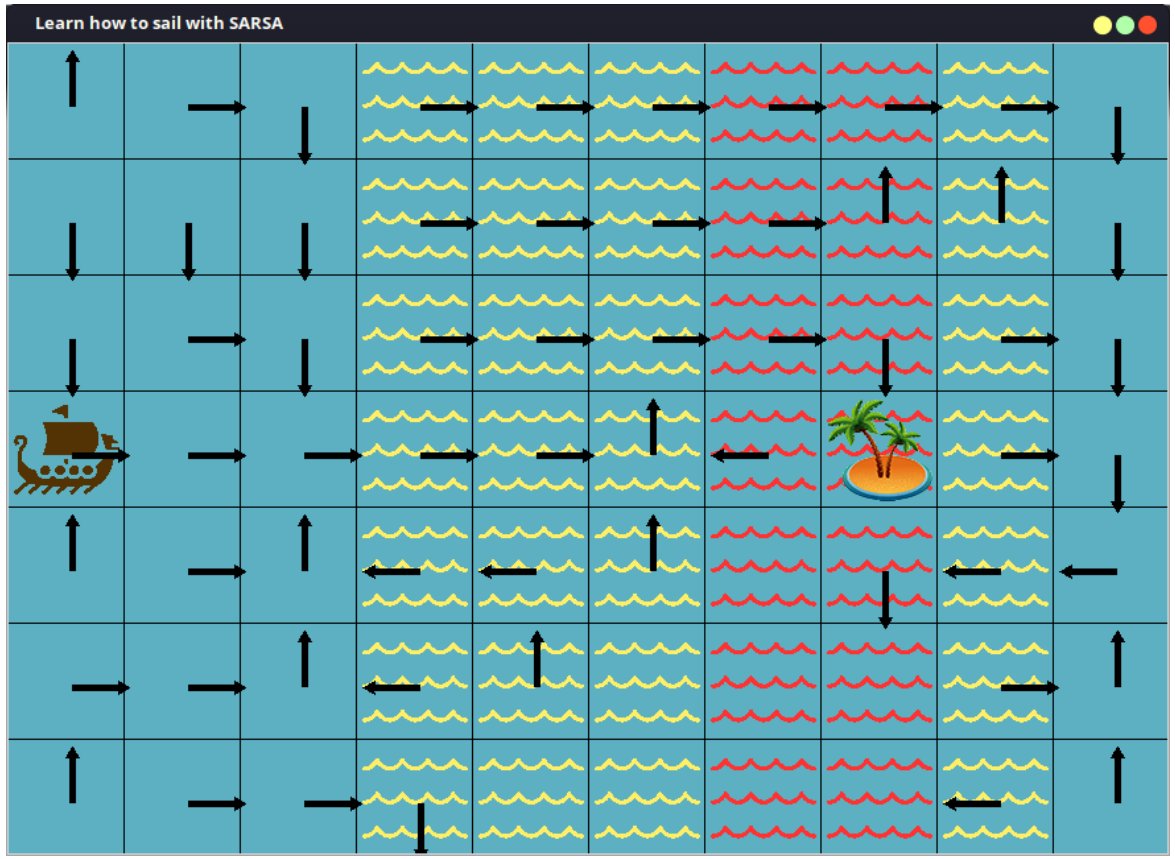


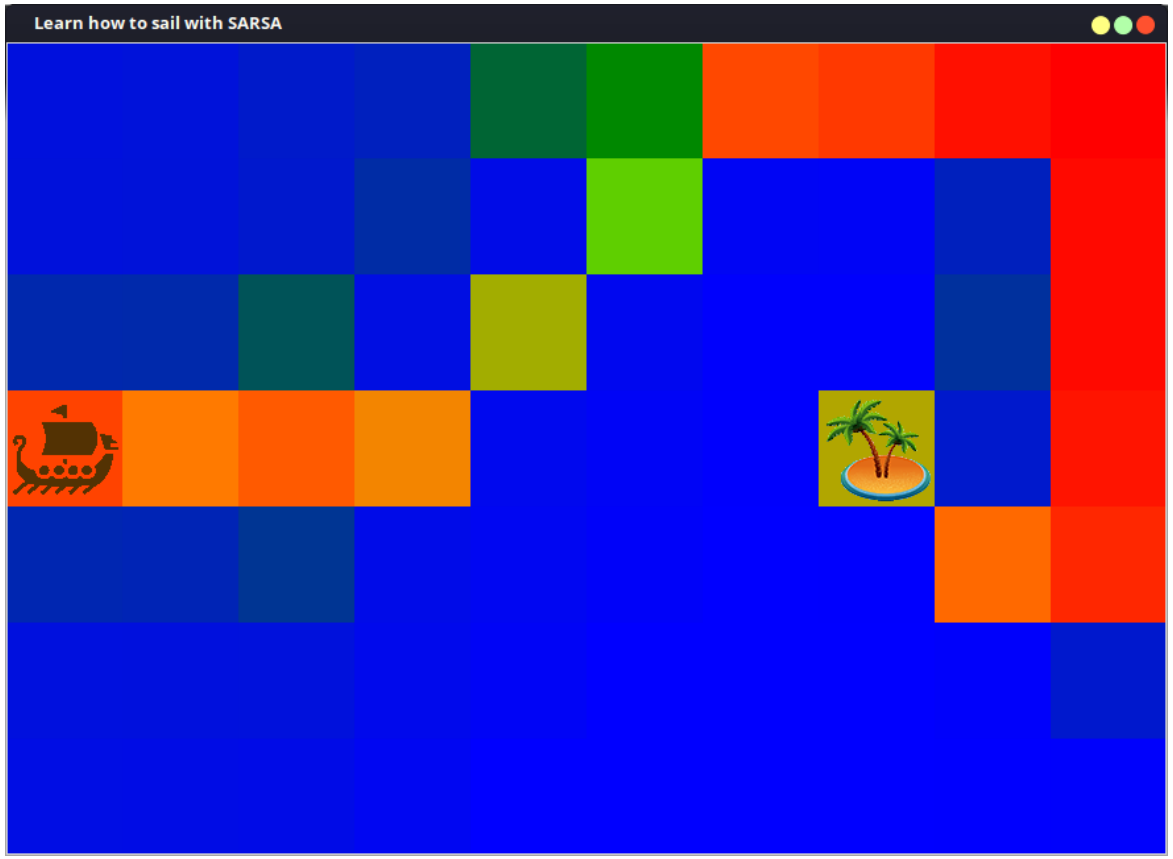
ERGEBNISSE

Aktionsmenge: Normal

Umgebung: Deterministisch

Beobachten von ε = 0.2



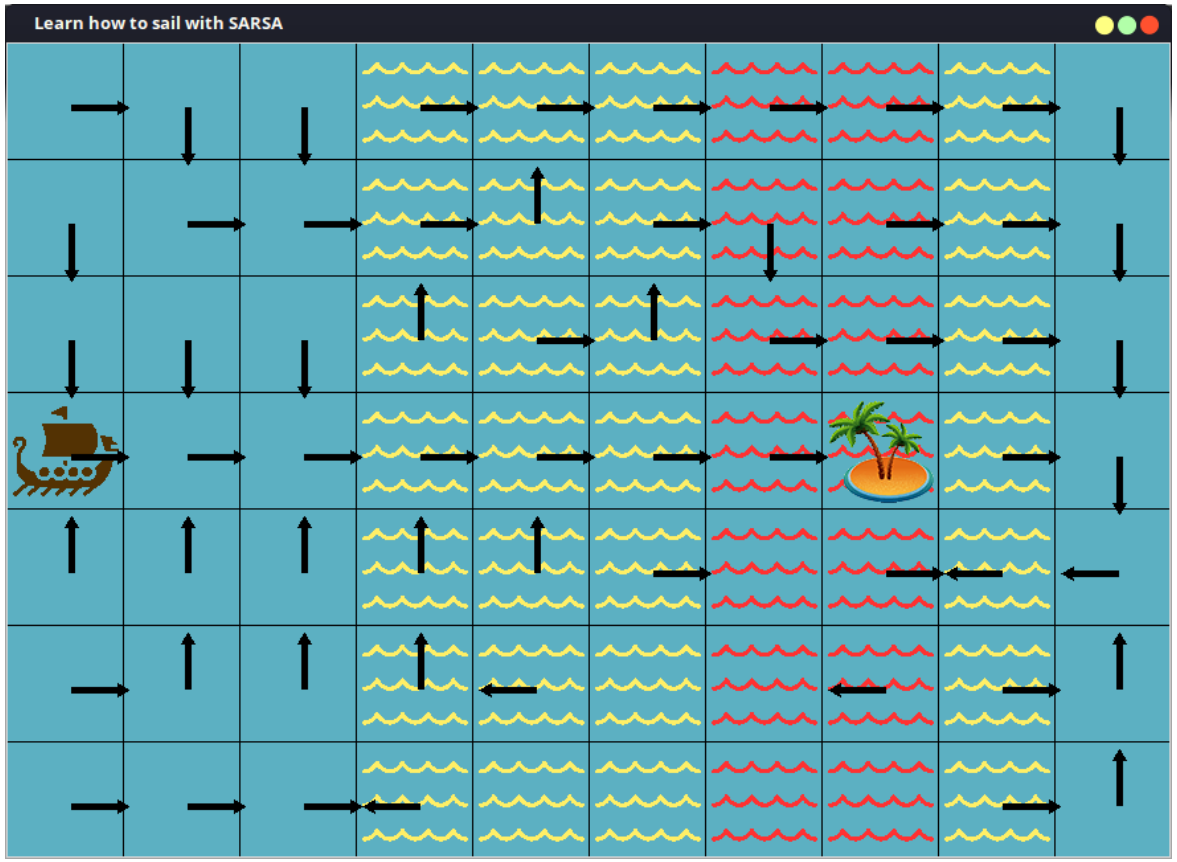


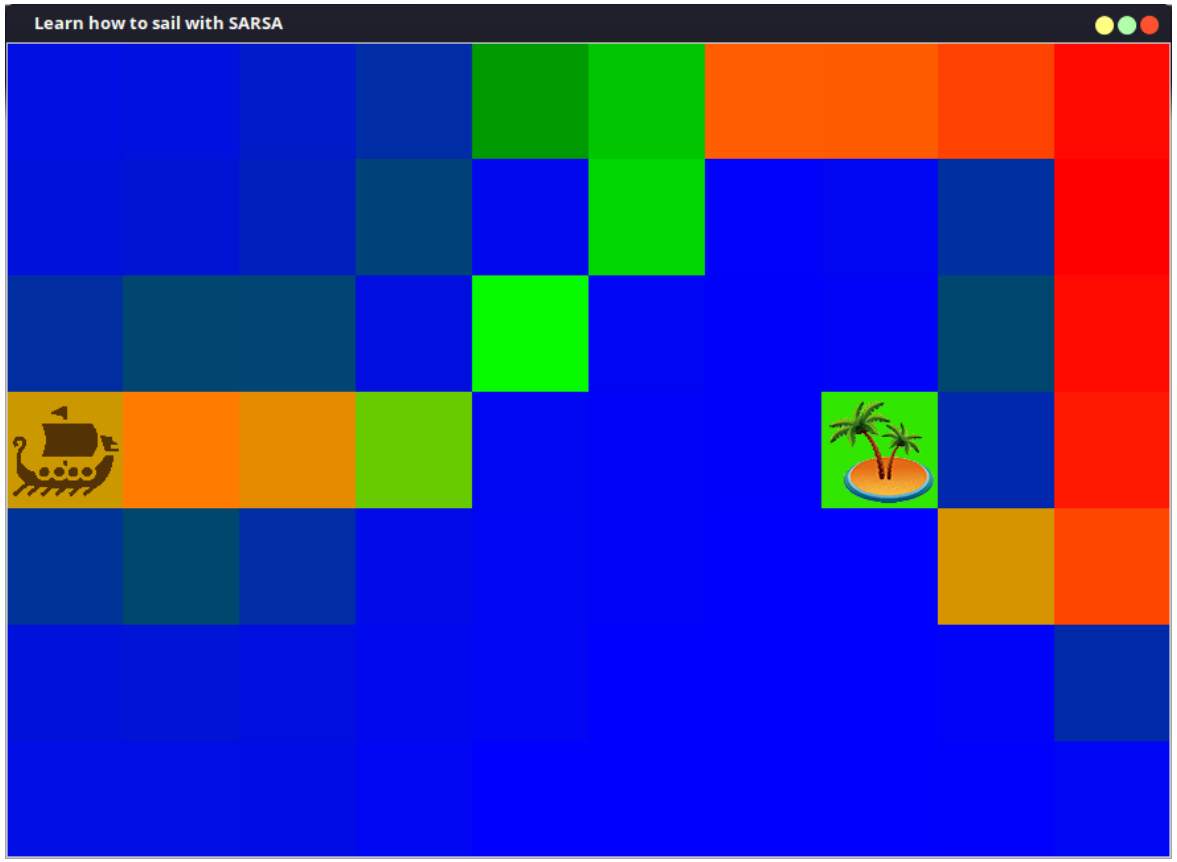
ERGEBNISSE

Aktionsmenge: Normal

Umgebung: Deterministisch

Beobachten von ε = 0.3



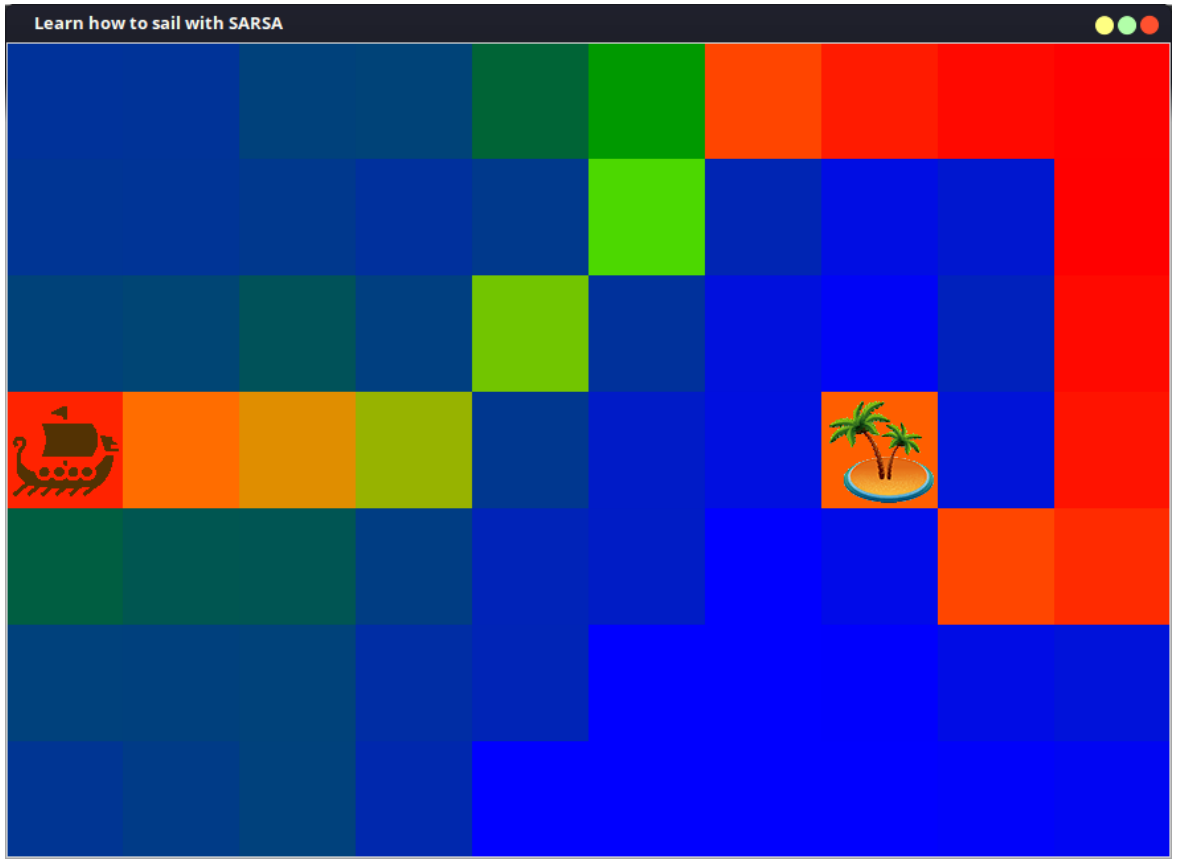


ERGEBNISSE

Aktionsmenge: Normal

Umgebung: Deterministisch

Beobachten von r = 0

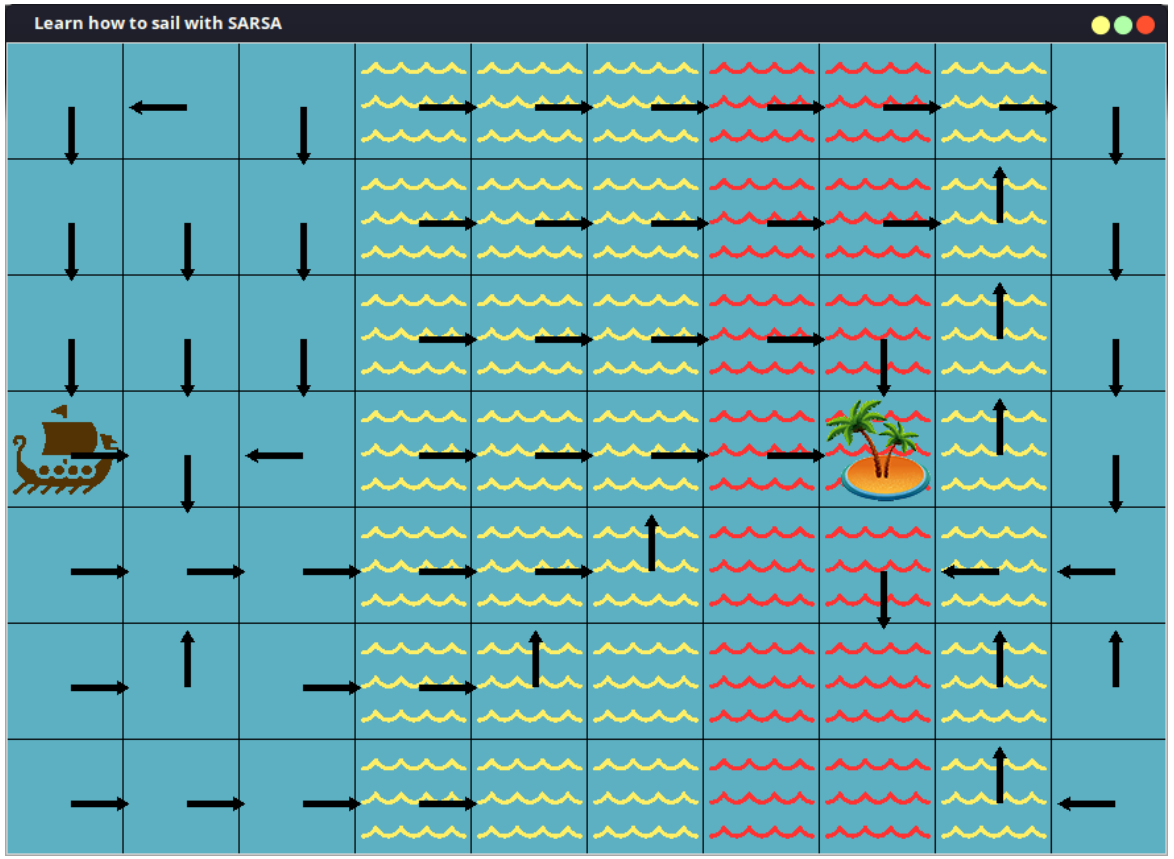


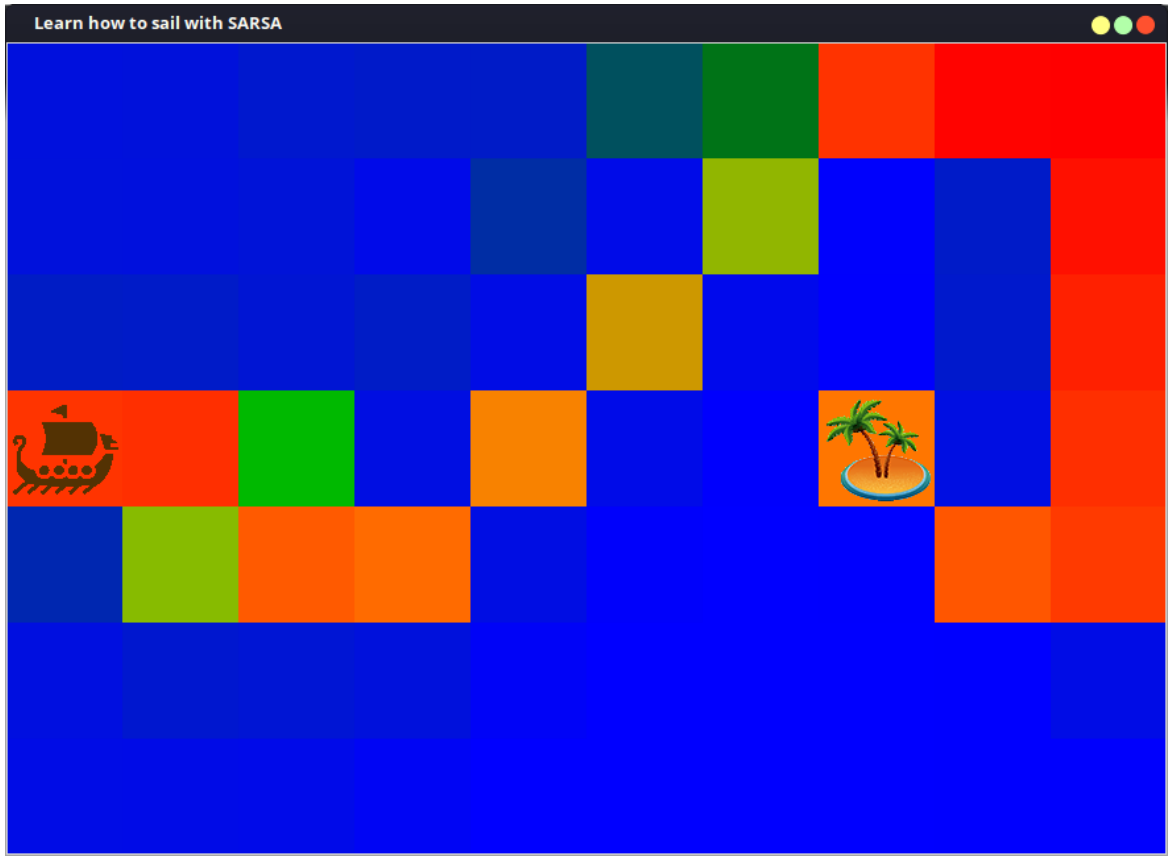
ERGEBNISSE

Aktionsmenge: Normal

Umgebung: Deterministisch

Beobachten von r = 100



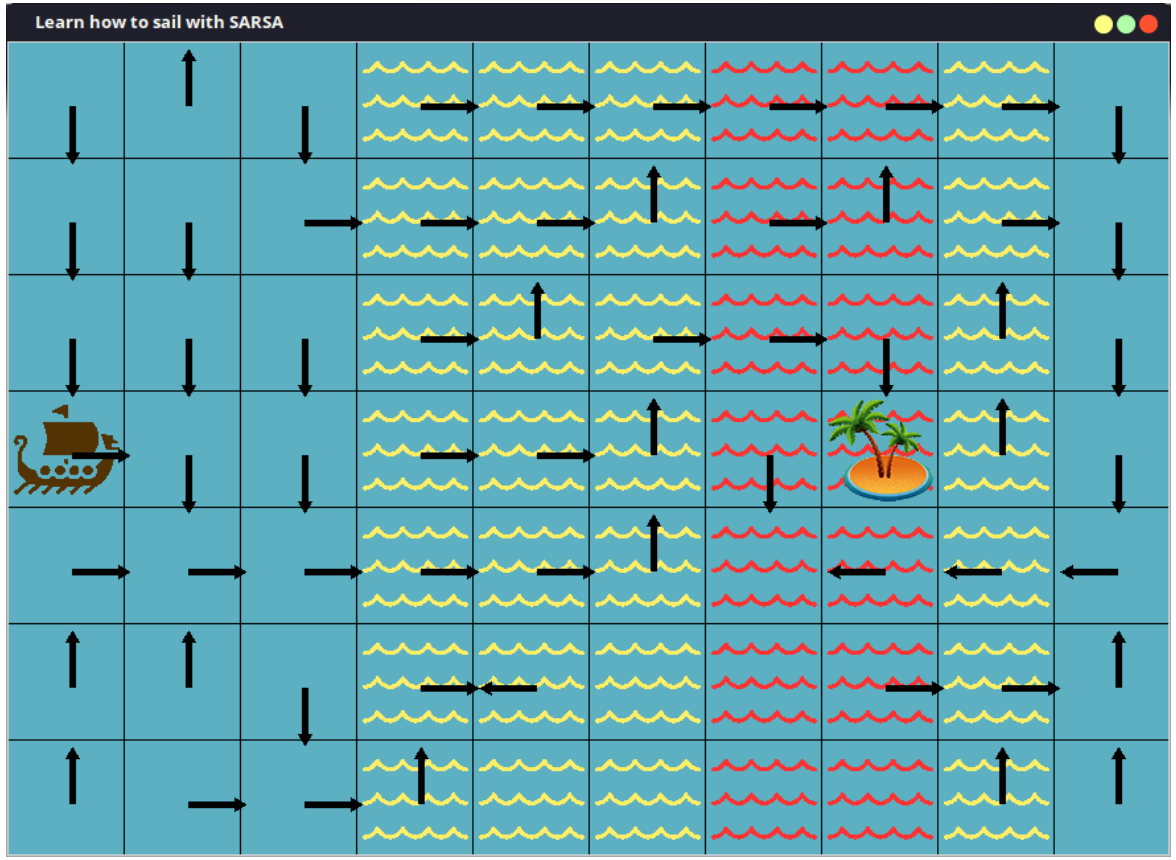


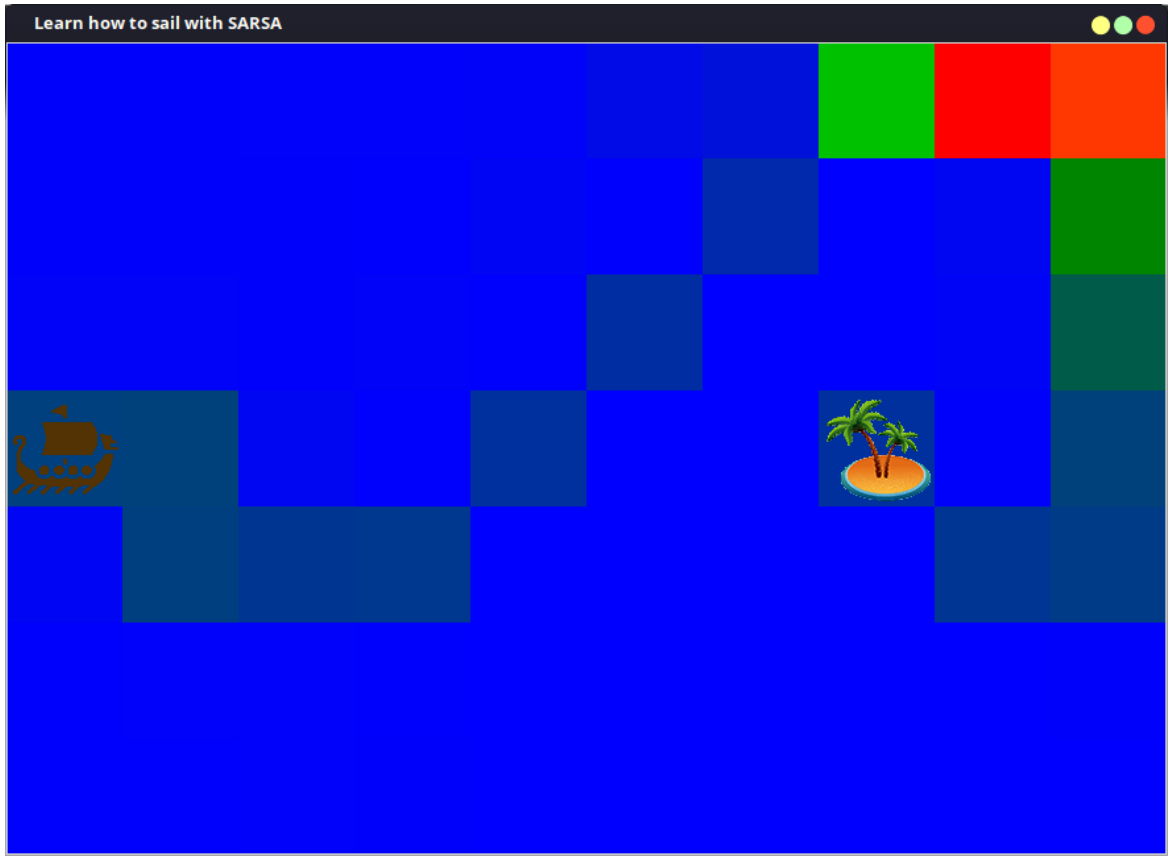
ERGEBNISSE

Aktionsmenge: Normal

Umgebung: Deterministisch

Beobachten von r = 10000





VIELEN DANK FÜR IHRE AUFMERKSAMKEIT

FRAGEN?

GRUNDLAGEN ADAPTIVER WISSENSSYSTEME

ALLGEMEINE INFORMATIK (M.SC.)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

FACULTY OF COMPUTER SCIENCE AND ENGINEERING